Fuzzy ART-Map for Learning Navigational Landmark Information

Verena Vanessa Hafner
Summer 1998
Universität Ulm, Germany

1 Scenario

A mobile robot (i.e. a Pioneer) equipped with a black and white ccd camera shall be guided by rectangular traffic signs that are posted along its way. The robot is moving in a maze made up of walls with vertical stripes. The signs indicate navigational information for the robot. Currently, five signs are supported: 'Move right', 'Move left', 'Move left or right', 'No robots', 'Cul-de-Sac'.

2 Project

The goal of this project was to find and classify the traffic signs in images taken by a camera. The project can be divided into two main parts:

- The preprocessing of the image which determines a region of interest (roi) (rectangular traffic sign) in the camera image. A feature vector taken from this area is returned.
- The learning and classification of traffic signs which is performed by a Simplified Fuzzy ART Map (sfam).

No robot motion commands are generated so far. The programming language that has been used for this project is C++.

3 Methods

A short description of the methods used for the task of recognizing signs in an image taken by a camera is given below.

3.1 Preprocessing of the images

The preprocessing of the images is preformed in the following stages:

1. Load the image either from the file system in pgm format or from the camera device in real time.
2. Determine the region of interest (roi), i.e. the traffic sign. This is done by extracting horizontal edges from the images with a sobel filter, thresholding them and performing a horizontal as well as a vertical projection.

These one-dimensional projections are then checked for plausible peaks which could indicate the borders of a traffic sign. If a traffic sign is indicated, the corresponding coordinates are returned.

3. Apply a sobel operator on the region of interest and split it into an $m * n$ grid of subregions, where m and n are small numbers (e.g. m=4 and n=3). The subregions are overlapping by one fourth of their length and height to smoothen the value change caused by small translations. To keep the dimension of the output feature vector relatively small, the following averaging method is used. The feature vector components v_i are simply calculated as
$$v_i = \frac{num}{totalnum},$$
where num is the number of pixels whose sobel value exceeds a certain threshold and $totalnum$ is the total number of pixels in the grid.[2]

3.2 Learning the categories

The main task of classifying the images is performed by a Simplified Fuzzy ART (Adaptive Resonance Theory) -Map (sfam). The sfam consists of an input layer, an output layer and weights connecting them. The number of input nodes is given by twice the dimension of the feature input vector since this vector is complement-coded. Complement coding is used to represent not only the presence of a feature in the input vector but also its absence. For example, if v was the feature input vector of dimension d and $\bar{v}$ its complement vector with $\bar{v}_i = 1 - v_i, (i = 1, ..., d)$, the input of the Fuzzy ART-Map input layer would be $I = (v, \bar{v}) = (v_1, ..., v_d, \bar{v}_1, ..., \bar{v}_d)$ of dimension 2d. Note that $|I| = \sum_{i=1}^{d} (v_i + \bar{v}_i) = d$, I is therefore automatically normalized.[3]

The output nodes are produced while learning, thus the final number of output nodes is not known in advance. Several output nodes will be assigned to one category. The category of the most active output node corresponding to a certain input is chosen. The output node with the highest activation wins.

The activation $T_j(I)$ of each output node j is given by

$$T_j(I) = \frac{|I \wedge w_j|}{\alpha + |w_j|},$$

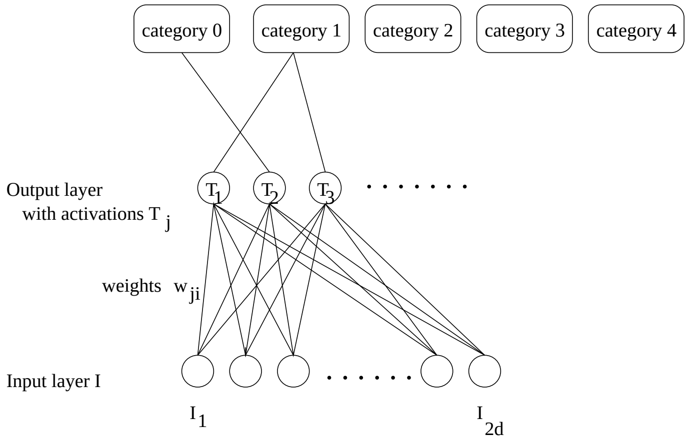
where I is the input vector, w_j is the weight vector for the j^{th} output node and α is the choice parameter, usually a small value close to 0 with $\alpha \neq 0$.
The output node with the highest activation wins. If the actual category does not match with the category of the winning output node, a new output node is created.

While learning, the weights w_j are adjusted according to the learning rule

$$w_j^{new} = \beta(I \wedge w_j^{old}) + (1 - \beta) w_j^{old},$$

where β is the learning rate which is set to 1 in this implementation of sfam for fast and simple learning.[1]

Figure 1: The Fuzzy ART Network



Resonance (which implies updating the weights of the winning output node) occurs if the match function of the winning output node meets the vigilance criterion

$$\frac{|I \wedge w_j|}{|I|} \geq \rho,$$

where ρ is the vigilance parameter that is fixed in advance to values between 0 and 1. If

$$\frac{|I \wedge w_j|}{|I|} < \rho,$$

a mismatch-reset occurs, the activation of j is set to 0 and a new winning node has to be chosen. If a new category comes up or no resonance occurs for an input vector, a new output node is created.

4 Software design and implementation

The project is mainly based on two C++ classes: vImage and sfam.
vImage can be used for the image processing, including loading and saving the image, applying filters onto it and extracting image related features.
sfam represents the Simplified Fuzzy ART-Map which has the ability of learning to categorize feature vectors.
The classes have been designed in this way to allow an independent representation of the feature extraction and the learning network. The resulting features from sfam can therefore be fed to several different neural nets or the Fuzzy ART-Map can learn to categorize feature vectors apart from images.

4.1 Class vImage

This class is used for preprocessing the camera image. At present, it contains two different sobel filters as well as methods to extract certain regions of interest. It also contains a method to calculate feature vectors of specified dimensions

from an indicated part of the image. vImage has its own image format to be independent of any standard pixel format. In this version, pgm files can be loaded and saved. New filters and file formats can be easily included into the class.
vImage provides the following public methods:

- int loadPgm(char[256])
  Syntax: loadpgm("./image.pgm")
- int savePgm(char[256])
  Syntax: savepgm("./image.pgm")
- int loadCam()
  not yet implemented
- int getPixel(int, int, int*)
  Syntax: getPixel(x,y,&value)
- int setPixel(int, int, int)
  Syntax: setPixel(x,y,value)
- void getSize(int*, int*)
  Syntax: getSize(&maxx,&maxy)
- int sobel(vImage*, int, int)
  Syntax: sobel(&image, hor, ver)
  This method applies a sobel operator onto the image. In this version, only the amplitude is used. *hor* and *ver* can be either 0 or 1.
- int sobelt(vImage*, int, int)
  This method works like the method 'sobel', but additionally applies a threshold, which is being calculated as twice the sum of all pixel values after applying the sobel operator divided by the total number of pixels.
- int getRoi(rect*)
  This method returns the region of interest in *rect* if one could be detected. Otherwise the errorcode 1 will be returned.
- void lineh(vImage, int*)
  Syntax: lineh(image, &line)
  This method performs a horizontal projection of the image. The result is returned in *line*.
- void linev(vImage, int*)
  Syntax: linev(image, &line)
  This method performs a vertical projection of the image. The result is returned in *line*.
- void rasta(int, int, rect*, float*)
  Syntax: rasta(x, y, &rectangle, &feature_vector)
  This method returns the (x*y)-dimensional feature vector v for a given region of the image. The feature vector components v_i are calculated as the number of pixels whose sobel value exceeds a certain threshold divided by the total number of pixels in the grid.

4.2 Class sfam

This class implements the Simplified Fuzzy ART Map. It can handle with feature vectors of any dimension and a variable number of categories. Parameters for public methods of sfam are the dimension of the feature vector, the vigilance ρ, the choice parameter α and the category for a feature vector to be learned. sfam provides the following public methods:

- void clearNet()
  This method deletes a current net and initializes for a new one.
- int loadNet(char*)
  Syntax: loadNet("./net")
- int saveNet(char*)
  Syntax: saveNet("./net")
- int train(int, float*, int, float, float)
  Syntax: num_output_nodes = train(dim, input_vector, category, alpha, vigilance)
- int check(int, float, float*)
  Syntax: category = check(dim, alpha, input_vector)

5 Example programs

For a brief introduction to the classes, there are five example programs provided, all performing different tasks:

- storefeatures <image.pgm> <cat> <filename>
  This program stores the feature vector and the category of an image into an ascii file. To store the features of a large number of images, it is useful to use a unix shell command similar to the following:

  ls *.pgm | while read F; do storefeatures $F 42 features.dat; done

- addtonet <image.pgm> <cat> <net.dat>
  This program adds the features and category of a given image to the net file. One should be aware of the fact that the performance of the net depends on the order and frequency an image feature input vector has been shown to the net.
- checknet <data1> <num1> <data2> <num2>
  This program trains a new net with the feature vectors of 'data 1' and checks it with the feature vectors of 'data 2'. The number of output nodes created and the hitrate is printed to standard out.
- checkcat <image.pgm> <net.dat>
  This program checks the category of a given image resulting from an earlier trained and saved net. The category is printed to standard out.

- xoutofy <data> <num> <outofnum>
  This program trains a new net with $n = (outofnum - num)$ randomly chosen features of data. The net is then checked with num features which has not been used for training. The hitrate is printed to standard out.

6 Sample runs and evaluation

6.1 Images and categories

For the sample runs described below, the following category numbers were attached to the traffic signs (see Figure 2).

image category	category number
Cul-de-Sac	0
Move left or right	1
No_robots	2
Move left	3
Move right	4

The images of the traffic signs have been taken by a black and white ccd-camera with varying angles α and distances d to the wall. Four sets of images have been taken:

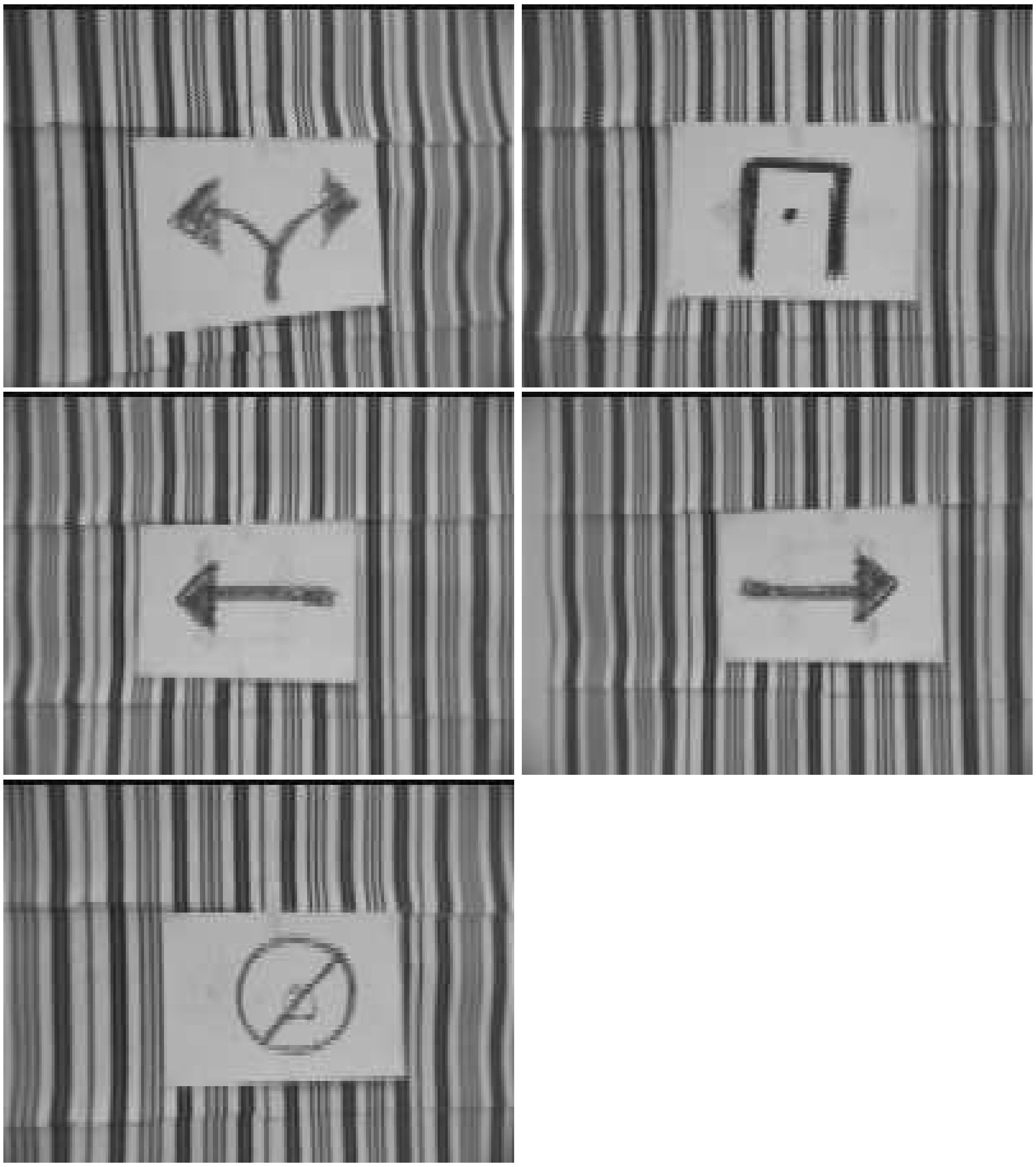
1. $-10° < \alpha < 10°, d \approx 100cm$
2. $-10° < \alpha < 10°, 70cm < d < 130cm$
3. $-50° < \alpha < 50°, d \approx 100cm$
4. $-50° < \alpha < 50°, 70cm < d < 130cm$

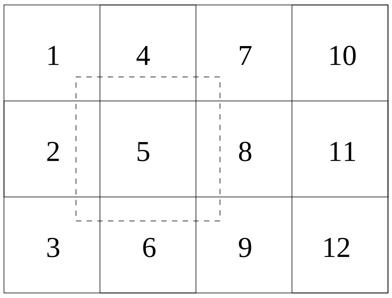
The method ‘getRoi’ of class vImage found a region of interest in 85% of the images. The missing 15% of the set of images were due to the fact that some traffic signs of the set of images were not laying completely in the image frame. Approximately 1% of the regions of interest found were incorrect.
Surprisingly, the learning success turned out best for set number 2, following 1, 3, 4. For the following results, a set of all four subsets has been used.

6.2 Extracted features

A typical feature vector for a traffic sign divided into a $4 * 3$ grid can look like the following:

Figure 2: sample images of each category



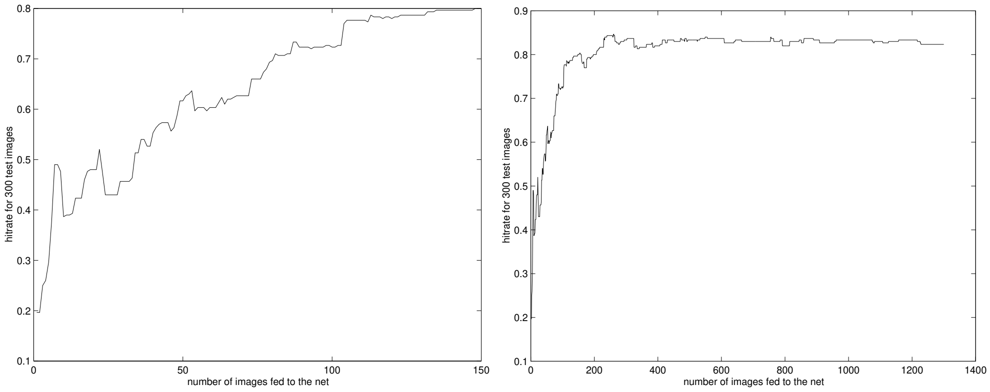


(0.367069, 0.283708, 0.226586, 0.163380, 0.122785, 0.122535, 0.263380, 0.250633, 0.197183, 0.141994, 0.146067, 0.163142)

6.3 Hitrates

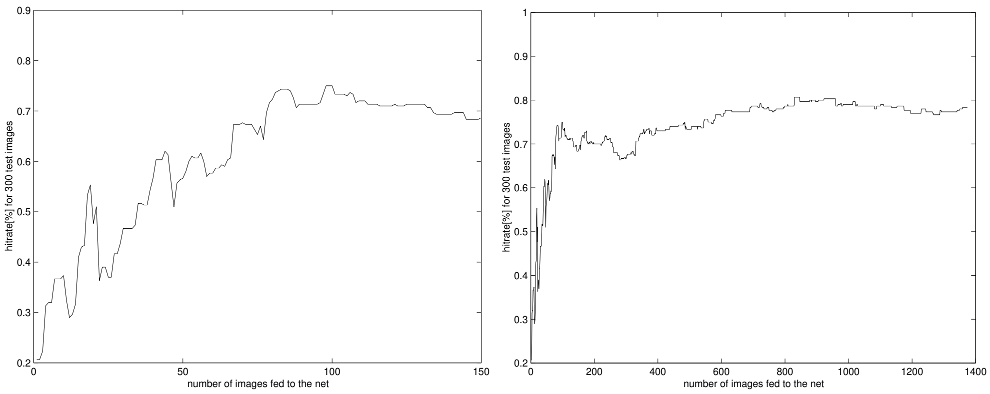
The hitrate for a certain number of learned images depends to a large extent on the order in which the images have been learned. Since a net can be saved and reused, it is valid to compare the best hitrates of different samples rather than the average hitrates.
The hitrates for a net trained with images and categories in random order has been higher than those for a net trained in sorted order. When a net was tested with the same images that has been used for training, the hitrate was always exactly 100%.

Figure 3: Hitrate for 300 test and 1300 training samples with 5 different categories. Number of grid tiles: 12, ρ: 0.4, α: 0.001



- Hitrate for 300 test and ca. 1400 training images with different grid tile numbers: (ρ: 0.4, α: 0.001)

Figure 4: Hitrate for 300 test and 1300 training samples with 5 different categories. Number of grid tiles: 120, ρ: 0.4, α: 0.001



	best hitrate[%]	mean hitrate[%]	worst hitr.[%]	std. dev.	#outputnodes
$4*3$	80	69.73	58	3.5180	50-70
$6*5$	90	81.07	71	3.1050	25-35
$12*10$	88	73.95	58	4.5874	15-30

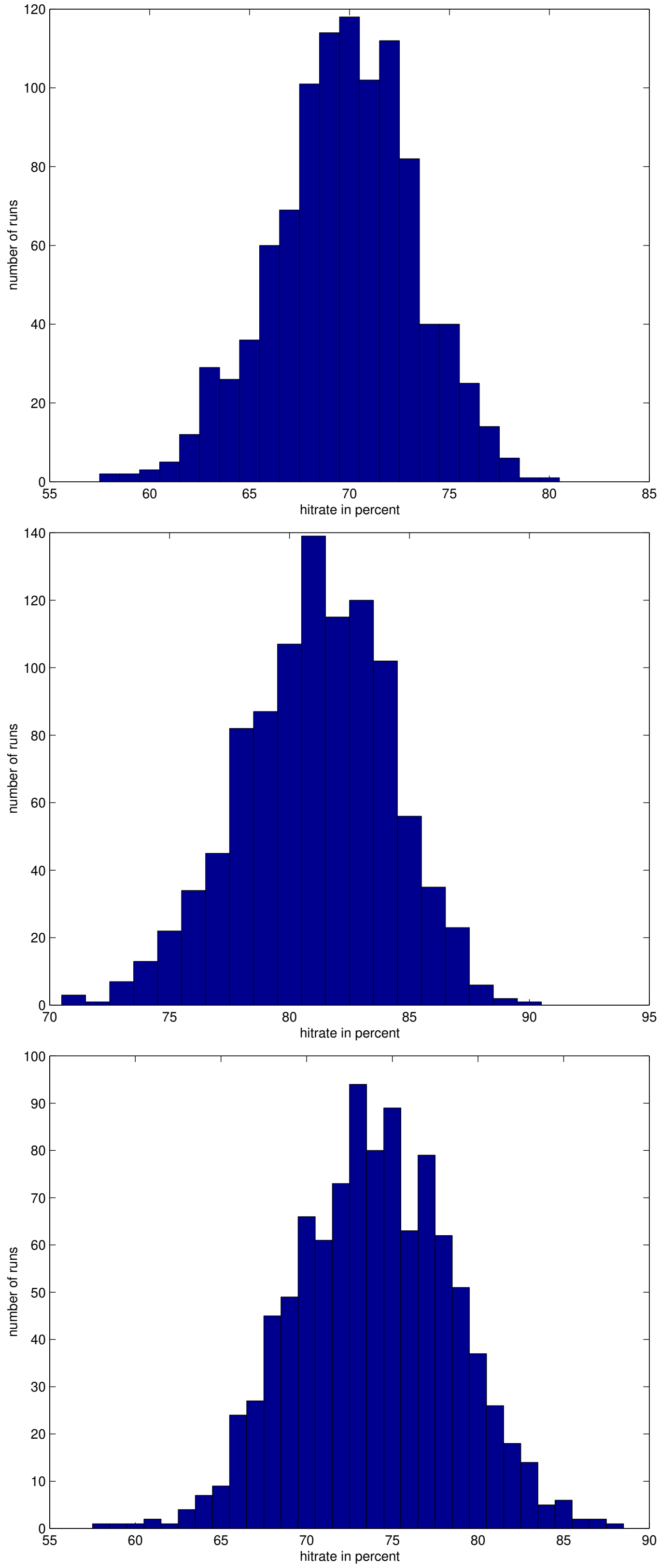
Note that the hitrate would be 20% if categories were arbitrarily chosen.

- Distribution of the classified categories (out of the categories 0, 1, 2, 3, 4) for a $4*3$ grid. 300 test images and ca. 1100 training images have been chosen randomly with equal category distribution fo 50 runs. The rows represent the image category, the columns the wrongly recognized category.

	0	1	2	3	4	Σ
0	1937	315	303	474	588	3617
1	36	2052	139	236	59	2522
2	222	176	2491	60	60	3009
3	320	374	47	1872	261	2874
4	485	83	20	358	2032	2978
Σ	3000	3000	3000	3000	3000	15000

- Distribution of the classified categories (out of the categories 0, 1, 2, 3, 4) for a $6*5$ grid. 300 test images and ca. 1100 training images have been chosen randomly with equal category distribution fo 50 runs. The rows represent the image category, the columns the wrongly recognized category.

Figure 5: Histograms of the hitrates



	0	1	2	3	4	Σ
0	2217	80	171	154	155	2777
1	48	2470	60	294	105	2977
2	168	61	2662	39	29	2959
3	142	235	59	2285	155	2876
4	425	154	48	228	2556	3411
Σ	3000	3000	3000	3000	3000	15000

- Distribution of the classified categories (out of the categories 0, 1, 2, 3, 4) for a $12 * 10$ grid. 300 test images and ca. 1100 training images have been chosen randomly with equal category distribution fo 50 runs. The rows represent the image category, the columns the wrongly recognized category.

	0	1	2	3	4	Σ
0	1982	46	90	308	141	2567
1	60	2393	72	233	161	2919
2	349	325	2748	249	255	3926
3	235	125	42	1960	149	2511
4	374	111	48	250	2294	3077
Σ	3000	3000	3000	3000	3000	15000

7 Limitations and extensions

7.1 Limitations

One limitation that could cause problems in future applications is the way how 'getRoi' (get region of interest) works. It was built to recognize rectangular posters on a vertically striped background. In this environment, the task is done very well. However, in a real world application, the robot could be set in front of cubes, plain walls or other robots and will eventually recognize them as posters.
The other limitation is the learning parameter of the Fuzzy ART-Map. It is set to 1, which gives very fast, quite good hitrates, however the hitrate does not converge.

7.2 Extensions

The main extension to this project is to use the classifying of images in "real world" robot navigation. A mobile robot shall be moving through the environment described in the scenario, avoiding walls. This will be done using an algorithm that incorporates simplified optical flow. At T-junctions and cul-de-sacs, traffic signs will be provided. The robot shall try to recognize and classify these signs using 'vimage' and 'sfam' and act according to some given rules (e.g. turn right, turn 180°, etc).
Extensions to the image recognition method could be an advanced filtering method that could, for example guarantee scale or rotation invariance (see [2]).

8 Conclusions

One conclusion that could be drawn from this project is, that Fuzzy ART-Map performs well and quickly in classification tasks. The number of output nodes and therefore the size of the net always remained reasonably small. The limitation of picture recognition was apparently mainly due to the image filtering methods.

References

[1] Tom Kasuba, *Simplified Fuzzy ARTMAP*, 1993, AI Expert 8:18-25, 1993

[2] Gary Bradski and Stephen Grossberg, *Fast-Learning VIEWNET Architectures for Recognizing Three-dimensional Objects from Multiple Two-dimensional Views* Neural Networks, Vol. 8, No. 7/8, pp. 1053-1080, 1995

[3] Gail A. Carpenter, Stephen Grossberg, Natalya Markuzon, John H. Reynolds, and David B. Rosen, *Fuzzy ARTMAP: A Neural Network Architecture for Incremental Supervised Learning of Analog Multidimensional Maps*, IEEE Transactions on Neural Networks, Vol.3, No.5, September 1992

9 Directory structure

All the files belonging to this project can be found within the directory
owl.informatik.uni-ulm.de:/home/vision/projects/verena
The subdirectories are:

doc:	documentation of the project
src:	the C++ source code
bin:	the executables to preprocess the images, train and test the net
img:	images that are used to train and test the system
data:	data files generated from the images, e.g. feature vector tables and sfam weight tables